一、事件是如何实现的?

// 常见事件对象的属性和方法

// 1. e.target 返回的是触发事件的对象（元素）， this 返回的是绑定事件的对象（元素）

// 区别： e.target 点击了那个元素，就返回那个元素， this 那个元素绑定了这个点击事件，那么就返回谁

事件基于发布订阅模式，就是在浏览器加载的时候会读取事件相关的代码，但是只有实际等到具体的事件触发的时候才会执行。

比如点击按钮，这是个事件 Event ，而负责处理事件的代码段通常被称为事件处理程序 Event Handler ，也就是「启动对话框的显示」这个动作。

在 Web 端，我们常见的就是 DOM 事件：

1. DOM0 级事件，直接在 html 元素上绑定 on-event ，比如 onclick ，取消的话， dom.onclick = null ，同一个事件只能有一个处理程序，后面的会覆盖前面的。
2. DOM2 级事件，通过 addEventListener 注册事件，通过 removeEventListener 来删除事件，一个事件可以有多个事件处理程序，按顺序执行，捕获事件和冒泡事件。
3. DOM3级事件，增加了事件类型，比如 UI 事件，焦点事件，鼠标事件。

- UI事件，即当用户与界面上的元素交互时触发。
- 焦点事件，即当用元素获得或失去焦点时触发。
- 鼠标事件，当用户通过鼠标在页面上执行操作时触发。

1.窗口事件

属性	描述
onblur	当窗口失去焦点时运行脚本。
onfocus	当窗口获得焦点时运行脚本。
onload	当文档加载之后运行脚本。
onresize	当调整窗口大小时运行脚本。
onstorage	当 Web Storage 区域更新时（存储空间中的数据发生变化时）运行脚本。

```
<script>
        //当窗口失去焦点时，输出“窗口失去焦点”
        window.onblur = function () {
            console.log("窗口失去焦点");
        };
        //当窗口获取焦点时，输出“窗口获取焦点”
```

```
        window.onfocus = function () {
            console.log("窗口获取焦点");
        };
        //当页面文档加载完成后，输出"Hello, World"
        window.onload = function () {
            console.log("Hello,World");
        };
        //当调整窗口大小时，输出"窗口大小正在改变"
        window.onresize = function () {
            console.log("窗口大小正在改变");
        };
    </script>
```

2.表单事件

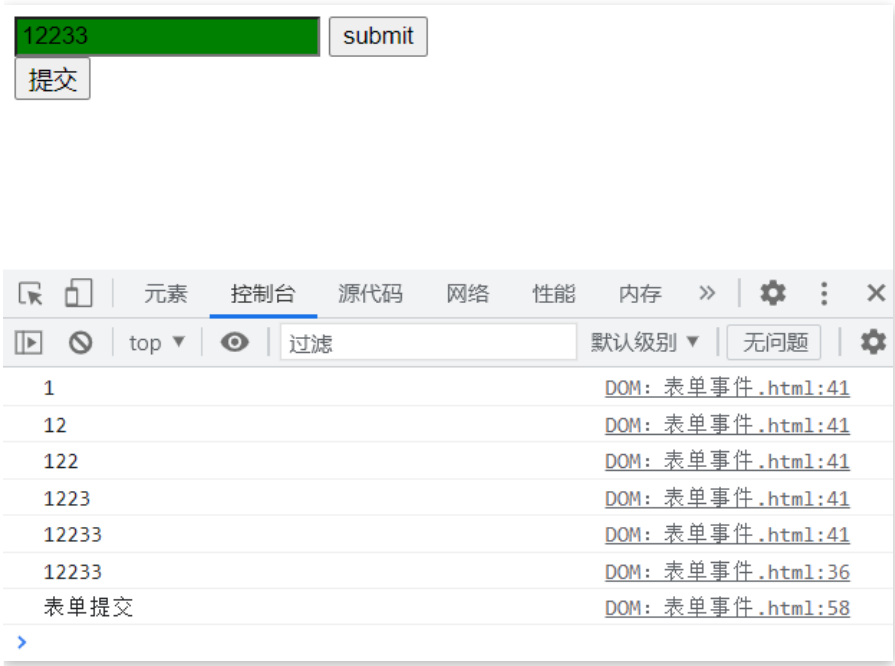
表单事件在HTML表单中触发 (适用于所有 HTML 元素，但该HTML元素需在form表单内)：

属性	描述
onblur	元素失去焦点时触发
onchange	该事件在表单元素的内容改变时触发 当输入文本变化且失去焦点(<input>, <select>, 和 <textarea>)
onfocus	元素获取焦点时触发
oninput	元素获取用户输入时触发
onreset	表单重置时触发
onsearch	用户向搜索域输入文本时触发 (<input="search">)
onselect	用户选取文本时触发 (<input> 和 <textarea>)
onsubmit	表单提交时触发

```
    <form>
        <input type="text" id="text" required>
        <input type="submit" value="submit">
    </form>
    <form id="myform">
```

```
        <input type="submit" id="submit">
    </form>
    <script>
        //1 当文本框获取焦点，文本框背景为红色，当文本框失去焦点，文本框背景为黄色
        var textInput = document.getElementById("text");

        /* 当文本框获取焦点，文本框背景为红色 */
        textInput.onfocus = function () {
            this.style.background = "red";
        };

        /* 当文本框失去焦点，文本框背景为绿色 */
        textInput.onblur = function () {
            this.style.background = "green";
        };


        //2 当文本框内容改变时，鼠标离开文本框，自动将文本框的内容输出到控制台
        textInput.onchange = function () {
            console.log(this.value);
        };

        //3 当文本框内容改变时，立即将改变的内容输出到控制台
        textInput.oninput = function () {
            console.log(this.value);
        };

        //4 如果单击“submit”，则不填写文本字段，将发生警报消息
        textInput.oninvalid = function () {
            console.log("请您完善表单内容！");
        };

        //5 当选中文本框的内容时，输出“您已经选择了文本框内容！”
        textInput.onselect = function () {
            console.log("您已经选择了文本框内容！");
        };

        //6 当提交表单的时候，在控制台输出“表单提交”
        let myform = document.getElementById("myform");

```

```
46          myform.onsubmit = function () {
47              console.log("表单提交");
48              return false; /* 用来阻止表单提交的，你不写它会跳转请求 */
49          };
```



3.键盘事件

属性	描述
onkeydown	当按下按键时运行脚本。
onkeyup	当松开按键时运行脚本。
onkeypress	当按下并松开按键时运行脚本。

4.鼠标事件

属性	描述
onclick	当单击鼠标时运行脚本。
ondblclick	当双击鼠标时运行脚本。
onmousedown	当按下鼠标按钮时运行脚本。
onmouseup	当松开鼠标按钮时运行脚本。
onmousemove	当鼠标指针移动时运行脚本。
onmouseover	当鼠标指针移至元素之上时运行脚本，不可以阻止冒泡。
onmouseout	当鼠标指针移出元素时运行脚本，不可以阻止冒泡。
onmouseenter	当鼠标指针移至元素之上时运行脚本，可以阻止冒泡。
onmouseleave	当鼠标指针移出元素时运行脚本，可以阻止冒泡。
onmousewheel	当转动鼠标滚轮时运行脚本。
onscroll	当滚动元素的滚动条时运行脚本。

mouseenter/mouseleave 与 mouseover/mouseout区别：

> `mouseleave` 和 `mouseout` 是相似的，但是两者的不同在于`mouseleave` 不会冒泡而`mouseout` 会冒泡。

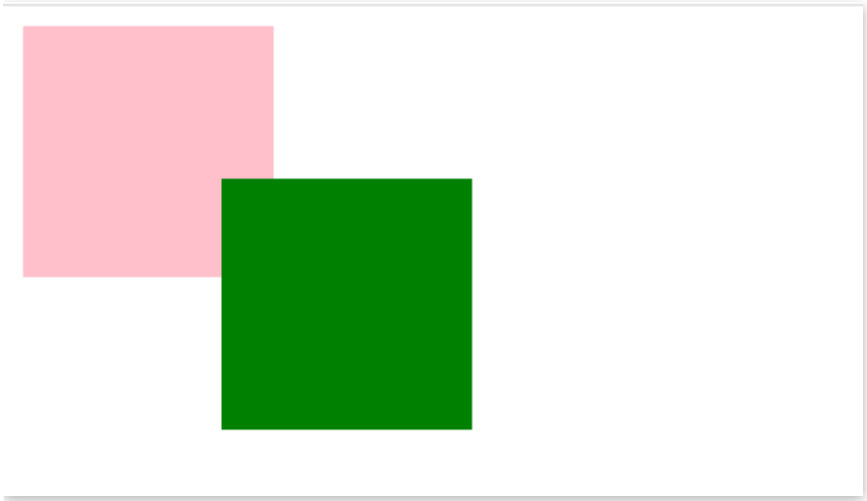
1

```
<div id="box" style="width: 100px;height: 100px;background: black;"></div>
    <script>
        let box = document.getElementById('box');
        /* 当鼠标移入div，背景颜色变为红色 */
        box.onmouseenter = function () {
            this.style.background = 'pink';
        }
        /* 当鼠标移出div，背景颜色变为绿色 */
        box.onmouseleave = function () {
            this.style.background = "green";
        };
    </script>
```

2

```
<div id="box1" style="width: 100px;height: 100px;background: pink;position: absolute;"></div>
    <div id="box2" style="width: 100px;height: 100px;background: green;position: absolute;"></div>

    <script>
        let box1 = document.getElementById('box1');
        let box2 = document.getElementById('box2');
        drag(box1);
        drag(box2);

        //提取一个专门用来设置拖拽的函数
        //参数：开启拖拽的元素
        function drag(obj) {
            //当鼠标在被拖拽元素上按下时，开始拖拽
            obj.onmousedown = function (event) {
                // 解决事件的兼容性问题
                event = event || window.event;

                /**
                设置obj捕获所有鼠标按下的事件：
                 setCapture():
                 只有IE支持，但是在火狐中调用时不会报错，
                 而如果使用chrome调用，它也会报错
                 */
                obj.setCapture && obj.setCapture();

                // obj的偏移量 鼠标.clentX - 元素.offsetLeft
                // obj的偏移量 鼠标.clentY - 元素.offsetTop
                let ol = event.clientX - obj.offsetLeft;
                let ot = event.clientY - obj.offsetTop;

```

```
                // 为document绑定一个鼠标移动事件
                document.onmouseover = function (event) {
                    // 解决事件的兼容性问题
                    event = event || window.event;

                    // 当鼠标移动时被拖拽元素跟随鼠标移动
                    // 获取鼠标的坐标
                    let left = event.clientX - ol;
                    let top = event.clientY - ot;

                    // 修改obj的位置
                    obj.style.left = left + 'px';
                    obj.style.top = top + 'px';
                }
                // 为document绑定一个鼠标松开事件
                document.onmouseup = function () {
                    // 取消document的onmousemove事件
                    document.onmousemove = null;
                    // 取消document的onmouseup事件
                    document.onmouseup = null;
                    // 当鼠标松开时，取消对事件的捕获
                    obj.releaseCapture && obj.releaseCapture();
                }
                /*
                  当我们拖拽一个网页中的内容时，浏览器会默认去搜索引擎中搜索内容，
                  此时会导致拖拽功能的异常，这个是浏览器提供的默认行为，
                  如果不希望发生这个行为，则可以通过return false来取消默认行为，
                  但是这招对IE8不起作用
                 */
                return false;
            }
        }
    </script>
```



5.媒体事件

属性	描述
onabort	当发生中止事件时运行脚本。
oncanplay	当媒介能够开始播放但可能因缓冲而需要停止时运行脚本。
oncanplaythrough	当媒介能够无需因缓冲而停止即可播放至结尾时运行脚本。
ondurationchange	当媒介长度改变时运行脚本。
onemptied	当媒介资源元素突然为空时（网络错误、加载错误等）运行脚本。
onended	当媒介已抵达结尾时运行脚本。
onerror	当在元素加载期间发生错误时运行脚本。
onloadeddata	当加载媒介数据时运行脚本。
onloadedmetadata	当媒介元素的持续时间以及其它媒介数据已加载时运行脚本。
onloadstart	当浏览器开始加载媒介数据时运行脚本。
onpause	当媒介数据暂停时运行脚本。
onplay	当媒介数据将要开始播放时运行脚本。
onplaying	当媒介数据已开始播放时运行脚本。
onprogress	当浏览器正在取媒介数据时运行脚本。
onratechange	当媒介数据的播放速率改变时运行脚本。
onreadystatechange	当就绪状态（ready-state）改变时运行脚本。
onseeked	当媒介元素的定位属性不再为真且定位已结束时运行脚本。
onseeking	当媒介元素的定位属性为真且定位已开始时运行脚本。
onstalled	当取回媒介数据过程中（延迟）存在错误时运行脚本。
onsuspend	当浏览器已在取媒介数据但在取回整个媒介文件之前停止时运行脚本。
ontimeupdate	当媒介改变其播放位置时运行脚本。
onvolumechange	当媒介改变音量亦或当音量被设置为静音时运行脚本。
onwaiting	当媒介已停止播放但打算继续播放时运行脚本。

6.其他事件

属性	描述
onshow	当 <menu> 元素在上下文显示时触发。
ontoggle	当用户打开或关闭 <details> 元素时触发。

7.事件冒泡

事件的冒泡（Bubble）：所谓的冒泡指的就是事件的向上传导，当后代元素上的事件被触发时，其祖先元素的相同事件也会被触发，在开发中大部分情况冒泡都是有用的，如果不希望发生事件冒泡可以通过事件对象来取消冒泡。

```
<style>
        #div1 {
            width: 200px;
            height: 200px;
            text-align: center;
            background: pink;
        }

        #div2 {
            width: 100px;
            height: 100px;
            text-align: center;
            margin: 0 auto;
            background: brown;
        }
    </style>

</head>

<body>
    <div id="div1">
        我是DIV1
        <div id="div2">
            我是DIV2
        </div>
    </div>
    <!-- --------------------------- -->
    <script>
        let div1 = document.getElementById('div1');
        let div2 = document.getElementById('div2');

        // 为div1绑定单击事件
        div1.onclick = function () {
            console.log('div1: 单击事件发生');
        }
```

```
        // 为div2绑定单击事件
        div2.onclick = function () {
            console.log('div2：单击事件发生')
        }
    </script>
</body>
```

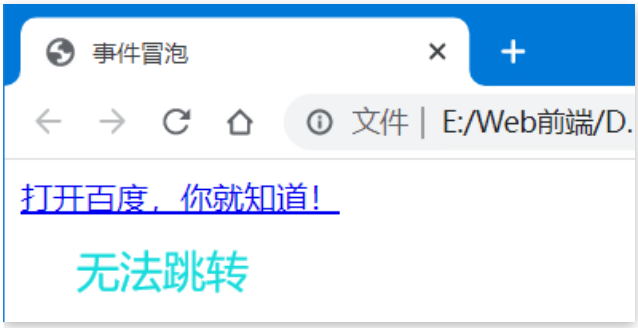
2 取消事件冒泡

```
<style>
        #div1 {
            width: 200px;
            height: 200px;
            text-align: center;
            background: pink;
        }

        #div2 {
            width: 100px;
            height: 100px;
            text-align: center;
            margin: 0 auto;
            background: brown;
```

```
        }
    </style>

</head>

<body>
    <div id="div1">
        我是DIV1
        <div id="div2">
            我是DIV2
        </div>
    </div>
    <!-- --------------------------- -->
    <script>
        let div1 = document.getElementById('div1');
        let div2 = document.getElementById('div2');

        // 为div1绑定单击事件
        div1.onclick = function () {
            console.log('div1：单击事件发生');
            stopBubble();
        }
        // 为div2绑定单击事件
        div2.onclick = function () {
            console.log('div2：单击事件发生')
            stopBubble();
        }

        // 取消事件冒泡
        function stopBubble(event) {
            // 如果提供了事件对象，则这是一个非IE浏览器
            if (event && event.stopPropagation) {
                // 因此它支持W3C的stopPropagation()方法
                event.stopPropagation();
            } else {
                // 否则，我们需要使用IE的方式来取消事件冒泡
                window.event.cancelBubble = true;
            }
        }
    </script>
```

3 阻止浏览器的默认行为

```
<a href="https://www.baidu.com" id="a">打开百度，你就知道！</a>
    <script>
        let a = document.getElementById('a');

        a.onclick = function () {
            stopDefault();
        }

        // 阻止浏览器的默认行为
        function stopDefault(event) {
            if (event && event.preventDefault) {
                // 阻止默认浏览器动作(W3C)
                event.preventDefault();
            } else {
                // IE中阻止函数器默认动作的方式
                window.event.returnValue = false;
            }
            return false;
        }
    </script>
```



4 阻止事件冒泡

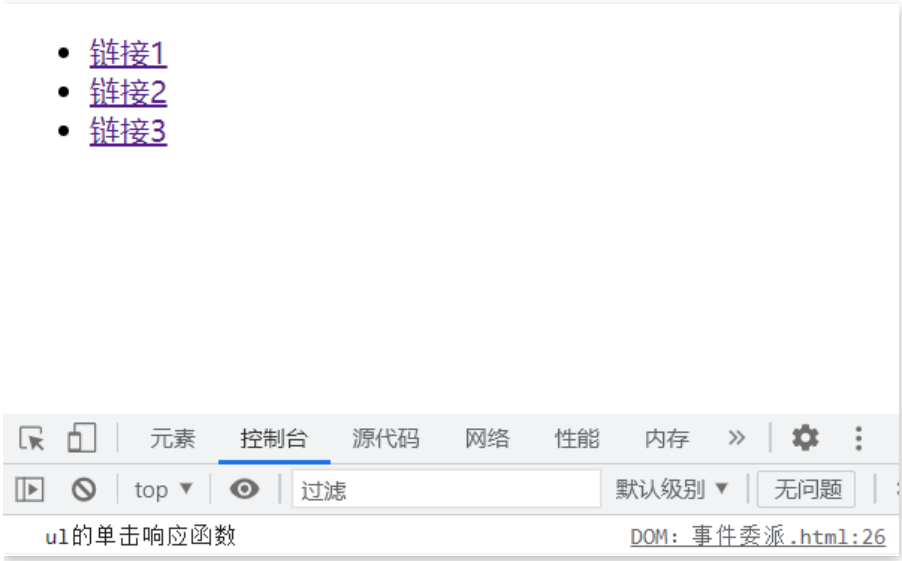
```
<body>
    <div class="demo">
        2222
        <div class="box">
            1111
            <div class="test">0000</div>
        </div>
    </div>

    <script>
        let demo = document.querySelector('.demo');
        let box = document.querySelector('.box');
        let test = document.querySelector('.test');
        /*
           eventPhase: 1(捕获)   2（目标）   3（冒泡）
       */
        demo.addEventListener('click', function (ev) {
            let e = ev || event;
            console.log('2222', e.eventPhase);
        })
        box.addEventListener('click', function (ev) {
            let e = ev || event;
            console.log('1111', e.eventPhase);
        })
        test.addEventListener('click', function (ev) {
            let e = ev || event;
            console.log('0000', e.eventPhase);
            // 阻止冒泡
            // e.stopPropagation();
            // 兼容
```

```
            e.stopPropagation ? e.stopPropagation() : e.cancelBubble = true;
        })
    </script>
</body>
```

8.事件委派

我们希望只绑定一次事件，即可应用到多个的元素上，即使元素是后添加的，我们可以尝试将其绑定给元素的共同的祖先元素，也就是事件的委派。事件的委派，是指将事件统一绑定给元素的共同的祖先元素，这样当后代元素上的事件触发时，会一直冒泡到祖先元素，从而通过祖先元素的响应函数来处理事件。事件委派原理是利用了事件冒泡，通过委派可以减少事件绑定的次数，提高程序的性能。

```
<ul id="ul">
        <li><a href="#" class="link">链接1</a></li>
        <li><a href="#" class="link">链接2</a></li>
        <li><a href="#" class="link">链接3</a></li>
    </ul>
    <script>
        let ul = document.getElementById('ul');

        // 为ul绑定一个单击响应函数
        ul.onclick = function (event) {
            event = event || window.event;
            // 如果触发事件的对象是我们期望的元素，则执行，否则不执行
            if (event.target.className == 'link') {
                console.log('ul的单击响应函数');
            }
        }
    </script>
```



eg1:

```
<body>
    <ul>
        <li>知否知否，点我应有弹框在手！</li>
        <li>知否知否，点我应有弹框在手！</li>
        <li>知否知否，点我应有弹框在手！</li>
        <li>知否知否，点我应有弹框在手！</li>
        <li>知否知否，点我应有弹框在手！</li>
    </ul>
    <script>
        // 事件委托的核心原理：给父节点添加侦听器，
        // 利用事件冒泡影响每一个子节点
        var ul = document.querySelector('ul');
        ul.addEventListener('click', function(e) {
            // alert('知否知否，点我应有弹框在手！');
            // e.target 这个可以得到我们点击的对象
            e.target.style.backgroundColor = 'pink';
        })
    </script>
</body>
```

eg2:

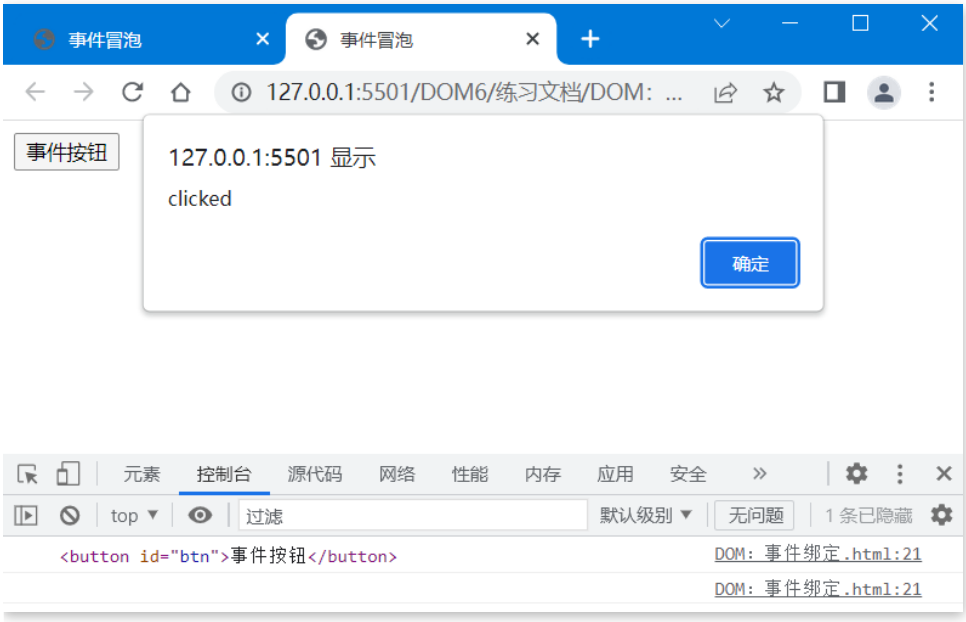
```
<ul id="ul">
        <li><a href="#" class="link">链接1</a></li>
        <li><a href="#" class="link">链接2</a></li>
        <li><a href="#" class="link">链接3</a></li>
```

```
        <li><a href="#" class="link">链接4</a></li>
        <li><a href="#" class="link">链接5</a></li>
    </ul>

    <script>
        // 事件委托
        let ul = document.getElementById('ul');

        ul.onclick = function (event) {
            console.log(this);
            console.log(event);
            event = event || window.event;
            if (event.target.className == 'link') {
                console.log('ul的单击响应函数');
            }
        }

    </script>
```

9.事件绑定

事件绑定可以理解为，在有一个触发事件的前提下，后面紧跟着一个事件处理函数，这个函数里面包含着所要执行动作的具体过程等，这就是事件绑定。

```
<button id="btn">事件按钮</button>
    <script>
        function bindEvent(elem, type, fn) {
            elem.addEventListener(type, fn);
        }

        const btn1 = document.getElementById('btn');
        bindEvent(btn1, 'click', event => {
            console.log(event.target); //event.target为获取触发的元素
            event.preventDefault(); //阻止默认行为
            alert('clicked');
        });
    </script>
```



10.事件传播

1. 捕获阶段：在捕获阶段时从最外层的祖先元素，向目标元素进行事件的捕获，但是默认此时不会触发事件
2. 目标阶段：事件捕获到目标元素，捕获结束开始在目标元素上触发事件
3. 冒泡阶段：事件从目标元素向它的祖先元素传递，依次触发祖先元素上的事件